Assignment2 - Technical report

Phoebe Zhou

September 2025

---

1. Executive Summary

This project compared two versions of a Retrieval-Augmented Generation (RAG) system: a straightforward baseline and a more advanced "enhanced" version. Both systems used dense retrieval to find relevant passages and then passed those passages to a language model to generate answers. The enhanced system added two extra steps — query rewriting and reranking — to improve the quality of retrieved context.

The results were clear. The enhanced system consistently outperformed the naive baseline. It produced more faithful answers, higher relevancy, and retrieved context that matched gold answers more often. Specifically, context precision and recall improved by about 15%, while answer faithfulness and relevancy improved by smaller but still meaningful margins.

Parameter experiment showed that using multiple passages via concatenation and instruction-style prompts gave the best balance of accuracy and efficiency. Persona and chain-of-thought prompts actually hurt performance. These findings highlight that better retrieval quality directly improves generated answers, and that not all prompting strategies are equally effective.

The system is modular, reproducible, and ready to be deployed at small or mid-scale. For production use, scalability and runtime cost remain the biggest challenges, especially for reranking.

2. System Architecture

The RAG system was designed to be modular and easy to extend. It starts with a corpus of around 3,200 passages and a gold evaluation set of 918 question–answer pairs. Passages are embedded using sentence-transformer models (such as all-MiniLM-L6-v2), and the embeddings are stored in an index.

The naive system takes each question as-is, retrieves the top-k passages from the index, and passes only the highest-ranked passage to the generator. This approach is lightweight and fast, but it risks missing better matches further down the ranking.

The enhanced system improves on this in two ways. First, it rewrites the query to remove stop words and normalize text, which helps with retrieval. Second, it applies a cross-encoder

reranker to rescore the top-10 retrieved passages and select the most relevant one. This extra computation makes the system slower, but it consistently improves context quality.

On the generation side, prompts can be formatted in different styles: instruction (direct Q&A), persona (as if from a tutor), or chain-of-thought (step-by-step reasoning). The generator defaults to Flan-T5, chosen for being lightweight, open-source, and cost-free. All predictions, contexts, and metrics are logged in JSONL or CSV formats for easy evaluation and reproducibility.

3. Experimental Results

The evaluation was done in three parts.

1. Prompt style comparison (Step 3):

When testing different prompt styles, instruction prompts were the clear winner, achieving F1 at 39 and EM at 32. Persona prompts dropped F1 to 26, and chain-of-thought collapsed F1 to 11. The takeaway here is that adding more elaborate prompting styles doesn't automatically help. In fact, for this setup it hurt accuracy.

2. Parameter Comparison (Step 4):

I tested different embedding models, retrieval depths (top-k), and passage selection strategies. Concatenating multiple passages worked best. For example, with all-MiniLM-L6-v2 and k=10 using concat, F1 reached at 42.6 and EM at 32.6, higher than first-passage selection, which plateaued around F1 at 39. Gains stabilized beyond k=10, which suggests that more context helps up to a point, but too much adds noise and length without much benefit. Different embedding models showed similar patterns, with small differences in F1 but the same general trend.

3. RAGAS evaluation between naive and enhanced (step 6):

Because of runtime limits, I ran 200 queries for this comparison. The enhanced pipeline consistently outperformed the naive one:

- Faithfulness: +0.08 (0.939 vs 0.856)
- Answer relevancy: +0.04 (0.775 vs 0.739)
- Context precision: +0.16 (0.854 vs 0.693)
- Context recall: +0.15 (0.805 vs 0.656)

These improvements show that better retrieval through query rewriting and reranking directly translates into better answers. Even modest gains in faithfulness and relevancy are meaningful in user-facing systems, while the large jump in context precision/recall confirms that the system is pulling in more useful evidence.

4. Enhancement Analysis

The enhancements I added addressed two weak spots in the naive pipeline: poor recall from raw queries and weak ranking of retrieved passages. Query rewriting cleaned up user inputs, making them more likely to hit relevant passages. Reranking then used a cross-encoder to pick the best among the retrieved candidates.

The results show that both steps made a real difference. Context alignment with gold answers improved sharply, and downstream answer quality followed. But the trade-off is speed. Reranking requires scoring every query-passage pair, which is several times slower than naive retrieval. Running this at scale without GPUs would be difficult.

Another surprise was the underperformance of chain-of-thought prompting. While it's often assumed to improve reasoning, in this setup it distracted the generator from producing concise, correct answers. Flan-T5 may simply not be strong enough to handle chain-of-thought effectively when paired with retrieved text.

Overall, the enhancements were worth the extra cost, especially when accuracy is more important than latency. In practice, a lighter reranker or hybrid retrieval strategy might be needed to keep the system responsive at scale.

5. Production Considerations

Scaling this system to larger corpora introduces new challenges. While indexing 3,200 passages is trivial, production workloads could involve millions. FAISS with GPU support or approximate nearest neighbor search would be required to keep retrieval fast.

Reranking improves answer quality but adds latency. To deploy at scale, it may be necessary to replace the cross-encoder with a faster bi-encoder or a distilled reranker model. Another option is to apply reranking selectively, only when confidence in the top-k retrieval is low.

Prompt style should stick with instruction format, which consistently gave the best results. Persona and chain-of-thought should be avoided unless specifically needed for user experience. The generator can remain Flan-T5 for cost control, though upgrading to a larger model may boost performance if accuracy becomes a higher priority.

Limitations of the current setup include reliance on English data, the need to subsample queries for evaluation due to runtime, and the lack of large-scale stress testing. For deployment, monitoring retrieval precision/recall and logging environment details (software versions, seeds) will be key to maintaining quality and reproducibility.

6. Appendices

1. AI Usage: All runs used HuggingFace models only — Flan-T5 for generation, MiniLM variants for embeddings, CrossEncoder reranker, and Phi-3.5-mini-instruct for RAGAS judging. No proprietary APIs were used. AI tools (including ChatGPT) were also used to assist with coding, debugging, and drafting documentation, though all experiments and results were run and verified locally.
2. Environment: Python 3.12, Torch, Transformers, SentenceTransformers, Pandas, FAISS (optional). GPU acceleration was used when available.
3. Reproducibility: For additional details, step-by-step outputs, and exploratory analysis, refer to the notebooks in the notebooks/ folder (system_evaluation.ipynb and final_analysis.ipynb). Ensure inputs (corpus.jsonl, gold.jsonl) are in the expected directories. Results are written to results/*.jsonl, *.json, and *.csv.